

Naval Research Laboratory

Stennis Space Center, MS 39529-5004



NRL/MR/7332--95-7681

Detecting Temperature Anomalies in Towed Sensor System Data

RICHARD K. MYRICK

*Ocean Sciences Branch
Oceanography Division*



July 14, 1995

19950822 007

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGEForm Approved
OBM No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE July 14, 1995	3. REPORT TYPE AND DATES COVERED Final	
4. TITLE AND SUBTITLE Detecting Temperature Anomalies in Towed Sensor System Data			5. FUNDING NUMBERS Job Order No. 5735136A5 Program Element No. 0603207N Project No. R0118 Task No. Accession No. DNN25501	
6. AUTHOR(S) Richard K. Myrick			8. PERFORMING ORGANIZATION REPORT NUMBER NRL/MR/7332--95-7681	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Research Laboratory Oceanography Division Stennis Space Center, MS 39529-5004			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) <p>Internal turbulent mixing events, internal gravity waves and encounters with fronts in the oceans are usually accompanied by large variances of water temperature and conductivity, relative to the average background state. Long chains of densely spaced sensors are towed through the water to detect and quantify these relatively rare, random events. The time and locations of the events cannot be predicted, so these sensors must collect data continuously in order that the events can be observed. This paper demonstrates a method for real-time analysis and sorting of oceanographic data from towed sensor chains. Temporal variations of the incoming data stream are calculated, displayed, and stored in near real-time.</p> <p style="text-align: right;">DTIC QUALITY INSPECTED 2</p>				
14. SUBJECT TERMS physical oceanography, optical oceanography, fine structure, instrumentation			15. NUMBER OF PAGES 32 16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT SAR	

Contents

Introduction	1
NRL Towed Sensor System	1
TSS Raw Data Format	2
TSS Relational Database Design and Implementation	2
Using Variance to Detect Temperature Anomalies in TSS Data	4
Acknowledgments	5
References	5
Figure 1. Temperature Data 09/14/94 (03:20 - 03:30)	6
Figure 2. Temperature Variance (5 Point Sample)	7
Figure 3. Temperature Variance (10 Point Sample)	8
Figure 4. Temperature Variance (25 Point Sample)	9
Appendix A. Software Documentation	10

Accession For	
NTIS CRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification _____	
By _____	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

DETECTING TEMPERATURE ANOMALIES IN TOWED SENSOR SYSTEM DATA

Introduction

The Towed Sensor System (TSS) is used extensively by the Meso & Finescale Ocean Physics Section (Code 7332) of the Naval Research Laboratory (NRL) for the collection of environmental data. This data is used for verification of oceanographic instrumentation systems, operational data for numerical models and to further the understanding of oceanographic processes.

The TSS is capable of collecting substantial amounts of information. A typical two week cruise can result in several gigabytes of unprocessed data. This large quantity of data, composed of samples from many different types of sensors, presents several problems, (1) extraction of data from specific sensors, (2) extraction of data from specific times and/or locations, (3) extraction of data within a range and (4) constant reformatting for post processing applications. The logical choice for the storage of TSS data is a relational database management system (RDBMS). A RDBMS allows concurrent access to the data by several researchers simultaneously while facilitating the extraction of data subsets for further examination.

Another difficulty in examining TSS data is identifying sections of data that are of interest, such as internal waves and fronts. These types of phenomena are typically represented in the data as variations from the mean temperature. Several methods of statistically "screening" the TSS data are examined to determine which gives the most sensitivity to temperature changes, yet does not detect common "noisy" data as areas that might contain useful data.

NRL Towed Sensor System

The Towed Sensor System (TSS) is a vertical sensor array consisting of an instrumented hard-faired cable, an Underwater Data Acquisition Module (UAM), dead-weight depressor, deck mounted hydraulic winch, and associated data recording and processing equipment. The instrumented (active) section of the array is contained in the lower 28 meters of the hard faired cable which is 300 meters length overall. The active section contains 38 temperature, 4 conductivity, 8 fluorescence, 2 irradiance, 2 backscatter and a pressure sensor spaced at user defined intervals. Other sensors can be added as necessary. Array depth is controlled by the amount of cable paid out from the deck mounted winch. A total of 300 m of hard-faired cable is stored as a single layer on the winch drum. This provides an operating depth of 200 m while towing at speeds up to 10 knots. The full sensor aperture can be used in depths as shallow as 50 m. The data is multiplexed and transmitted from the UAM, located in the depressor, up the cable to computers on the surface vessel for recording, display, and processing.

The TSS is composed of two sub-systems: data collection and data processing. Data collection instrumentation consists of the instrumented hard-faired cable, UAM, deck mounted winch, and ship board sensors (Global Positioning System and surface irradiance). Data processing instrumentation consists of a suite of PC-based computers for real-time data processing, graphical displays, and data storage. The TSS collects approximately 1 megabyte of data every 10 minutes. Normally, the TSS is towed at 5-7 knots for 6-12 hours. However, it is common practice to make tows lasting up to several days, stopping data collection occasionally to archive and purge hard disks

to make room for more data. Post processing of TSS data is done between tows, when possible, and upon return to port.

TSS Raw Data Format

Raw (unprocessed) TSS data is stored as an ASCII file. Each line of the file contains the data from the environmental sensors combined with the position and time information obtained from the Global Positioning System (GPS) navigation unit. Each line of TSS data contains a 1 second average of data from each sensor. Following is an example of several lines from a raw data file. The first line contain the "header" information which identifies the location and sensor type of each line in the file, the next 2 lines contain typical TSS data.

Date, Time, Longitude, Latitude, Press, Temp00, Temp01, Temp02, Temp03, Temp04, Temp05, Temp29, Temp06, Temp30, Temp07, Temp31, Temp08, Temp32, Temp09, Temp33, Temp10, Temp11, Temp12, Temp13, Temp14, Temp15, Temp16, Temp17, Temp18, Temp19, Temp20, Temp21, Temp34, Temp22, Temp35, Temp23, Temp36, Temp24, Temp37, Temp25, Temp38, Temp26, Temp27, Temp28, Cond01, Cond02, Cond03, Bkscat01, Bkscat02, Fluor01, Fluor02, Fluor03, Fluor04, Fluor05, Fluor06, Fluor07, Fluor08, Irr01, Irr02, Irr03, Irr04, Gmux, Vref, Roll, Pitch, Tnsn, Clamp, C28volt, SL410, SL488, SL550, SL683

```
09/13/94 19:54:58 18.52762 39.57382 47.37882 -5.052568 7.687677 21.19702 21.74078
-1.22375 22.68844 22.95005 28.47875 23.36997 -931226.8 22.90799 -55.20429 22.88253
23.27654 19.86534 23.09253 -15.07483 23.36888 23.76168 23.33391 23.62404 23.35422
23.3887 23.38645 23.55791 23.44575 23.46681 28.67078 23.29042 23.24727 .3036568
23.21437 23.44384 23.04844 20.93733 23.06724 16.56498 1.905851 1.273594 52.95577
40.95821 42.0886 1.077598E-03 -2.842303E-03 0 -2.577176 -3.274031 -4.435547 -1.50453
-6.402646 -1.913592 -4.781524 0 -.054073 .245425 0 2.2925E-04 -2.609994 .2631749
3.190445 887.8455 4.01945 4.055232 0 0 0 0
```

```
09/13/94 19:54:59 18.52762 39.57382 47.76311 -7.815691 9.294282 22.49779 23.4804
-2.960411 23.64264 24.2822 30.72789 24.53637 -987839.4 24.13439 -53.69165 24.26136
24.39591 20.68912 24.12177 -18.33576 24.48707 24.81461 24.43919 24.63046 24.45146
24.53035 24.52551 24.61691 24.54172 24.57451 28.91345 24.46404 24.5171 -2.184001
24.50593 24.51696 24.39027 20.77246 24.40246 17.17164 -.509811 -1.341887 61.68029
47.66944 49.11829 2.005003E-03 -2.842275E-03 0 -2.577575 -3.274461 -4.435841
-1.504729 -6.403085 -1.913764 -4.781841 0 -.1095093 .329125 0 2.2925E-04 -2.621668
.1991683 3.03735 910.6711 4.00545 4.056796 0 0 0
```

TSS Relational Database Design and Implementation

INGRES (Interactive Graphics and Retrieval System) version 8.9 operating under the Linux version 1.01 of the UNIX operating system was chosen as the RDBMS. INGRES is a system based on the relational model that allows any number of users (end-users or application programmers or both) to access any number of relational databases by means of the INGRES relational language

QUEL (Query Language). QUEL statements can be embedded into "C" programs using EQUEL (Embedded Query Language).

The TSS processing software PARTS (Processing And Retrieval of Towed Sensor, code listing in Appendix A.) is designed to read the raw TSS data files and create the TSS INGRES database. A user defined database is created containing a relation representing each sensor type found in the TSS data. The relations created are:

- Position - Date , Time, Latitude, and Longitude.
- Temp - Date, Time, Sensor ID, Temperature, and Sensor Depth
- Cond - Date, Time, Sensor ID, Conductivity, and Sensor Depth
- Optics - Date, Time, Sensor ID, Irradiance or Backscatter, and Sensor Depth
- Fluor - Date, Time, Sensor ID, Florescence, and Sensor Depth
- Ref - Date, Time, Reference ID, Reference values for the following:
 - Gmux - Multiplexer noise level
 - Vref - DC voltage
 - Roll - Depressor Roll
 - Pitch - Depressor Pitch
 - Tnsn - Cable Tension
 - Clamp - Lamp Voltage
 - C28 - 28 Volt DC Current

Date and time are used as primary indexes into each relation. A typical operation on the database would be to extract to data to create a file, for example:

Extract the position, temperature and depth data between the hours of 0900 and 1200 on November 14, 1994 with temperature values between 15.0 and 19.0 and store the results in a file called "11-14-94.temp.data" in the following format:

Date Time Latitude Longitude Temperature Depth

These QUEL statements will create the temporary relation tempdata and store it in the specified file.

range of p is position

range of t is temp

retrieve into tempdata (p.date, p.time, p.lat, p.lon, t.temp, t.depth)

where p.date = "11/14/94" and

p.time > "09:00:00" and

p.time < "12:00:00" and

p.date = t.date and

p.time = t.time and

t.temp > 15.0 and

t.temp < 19.0

copy tempdata

(date=c0,sp=d1,time=c0,sp=d1,lat=c0,sp=d1,lon=c0,sp=d1,temp=c0,sp=d1,depth=c0,nl=d1)

into "~/11-14-94.temp.data"

destroy tempdata

* c0 denotes a field of variable width, sp indicates a space delimiter and nl indicates a new line delimiter.

Using Variance to Detect Temperature Anomalies in TSS Data

In an effort to reduce the amount of time involved in processing TSS data "variance screening" can be used to detect temperature anomalies which may indicate the passage of fronts or internal waves. The program gettempvar (code listing in Appendix B.) was created to produce data files which, when viewed graphically using Microsoft Excel or similar programs, easily enable the user to identify whether anomalies exist in a particular data subset.

The user is asked to specify a date and time range to examine. The query is then performed on the specified INGRES database to extract the requested data and the variance operation is performed on the resulting data. The standard statistical formula is used to calculate the variance:

$$\text{Var} (x_1 \dots x_n) = (1/n-1) \left[n \sum_{j=1}^n (x_j - \bar{x})^2 \right]$$

The sample size (n) used to calculate the variance can be changed by altering the SAMPLE_SIZE variable in the header file var.h. A sample data subset was extracted for use in testing the effect of changing the sample size. The sample data is from September 14, 1994 between 03:20:00 and 03:30:00 (figure 1.). Several different sample sizes ranging from 2 to 25 were examined in order to determine which gives the best result for detecting the temperature anomalies. Results using sample sizes of 5, 10 and 25 are shown in figures 2-4.

A number of conclusions can be drawn from these tests. Smaller sample sizes tend to enhance abrupt changes in the data, but also can be misinterpreted if a channel is noisy. Larger sample sizes lessen noise but subtle temperature changes are spread over time and reduced to the point of being overlooked. A sample size of 10 gives the best overall result.

Acknowledgments

This work was supported by the Office of Naval Research (ONR) under Program Element 0603207N, "Ocean Measurement System."

References

Richard Myrick, Bruce Bricker and Mike Wilcox, "The Towed Sensor System Operations Manual", Naval Research Laboratory Code 7332, February 1992.

Rachel Morgan and Henry Morgan, *Introducing UNIX System IV*, McGraw-Hill Inc., New York, NY, 1987.

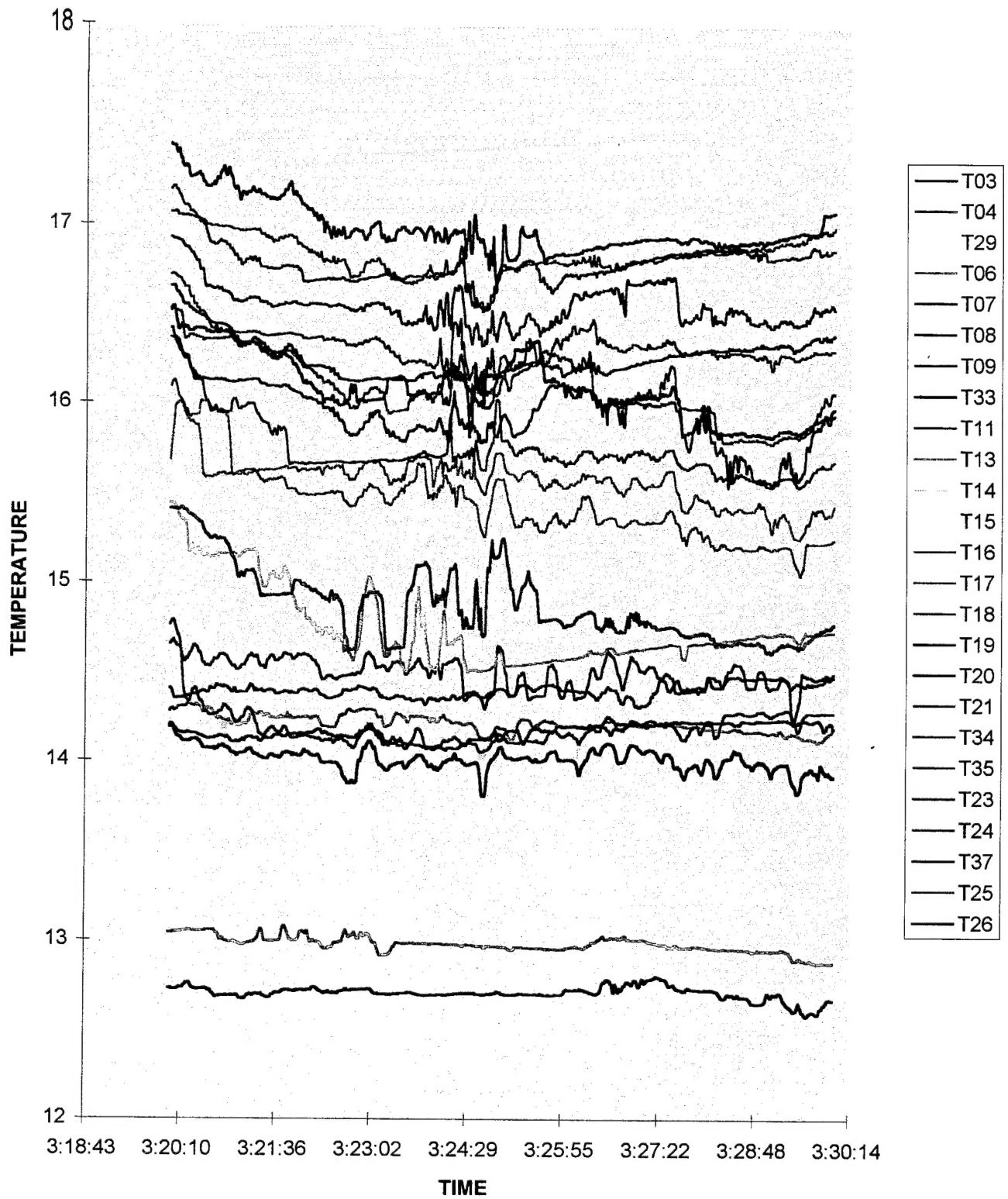
Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, Prentice-Hall Software Series, Prentice-Hall Inc., Englewood Cliffs, NJ, 1978.

C. J. Date, *A Guide to INGRES*, Addison-Wesley Publishing Company, Reading, MA, 1989.

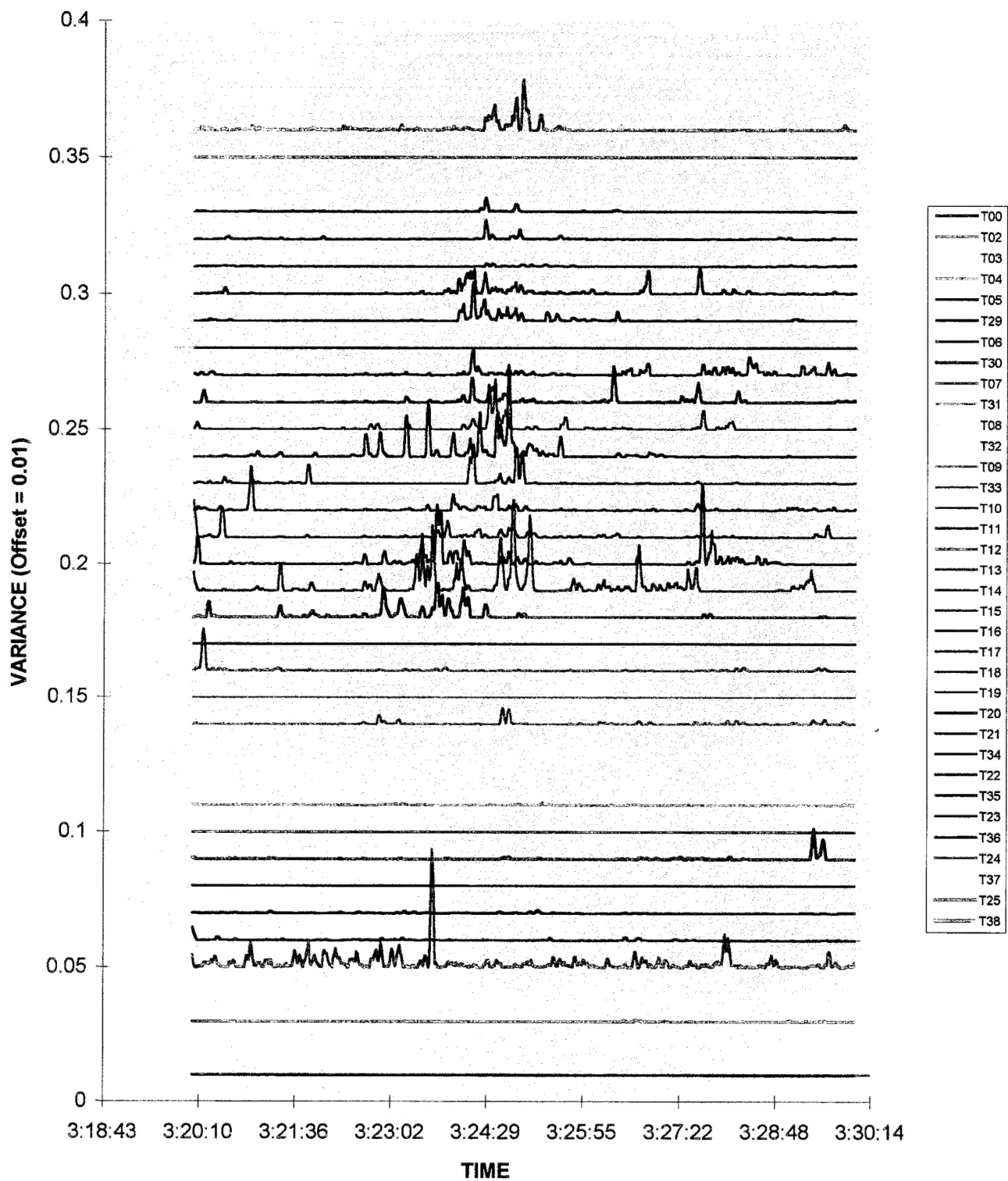
William H. Press, Brian P. Flannery, Saul A. Teukolsky and William T. Vetterling, *Numerical Recipes in C*, Cambridge University Press, Cambridge, England, 1988.

Robert Epstein, "Creating and Maintaining a Database Using INGRES", Electronics Research Laboratory, December, 1977.

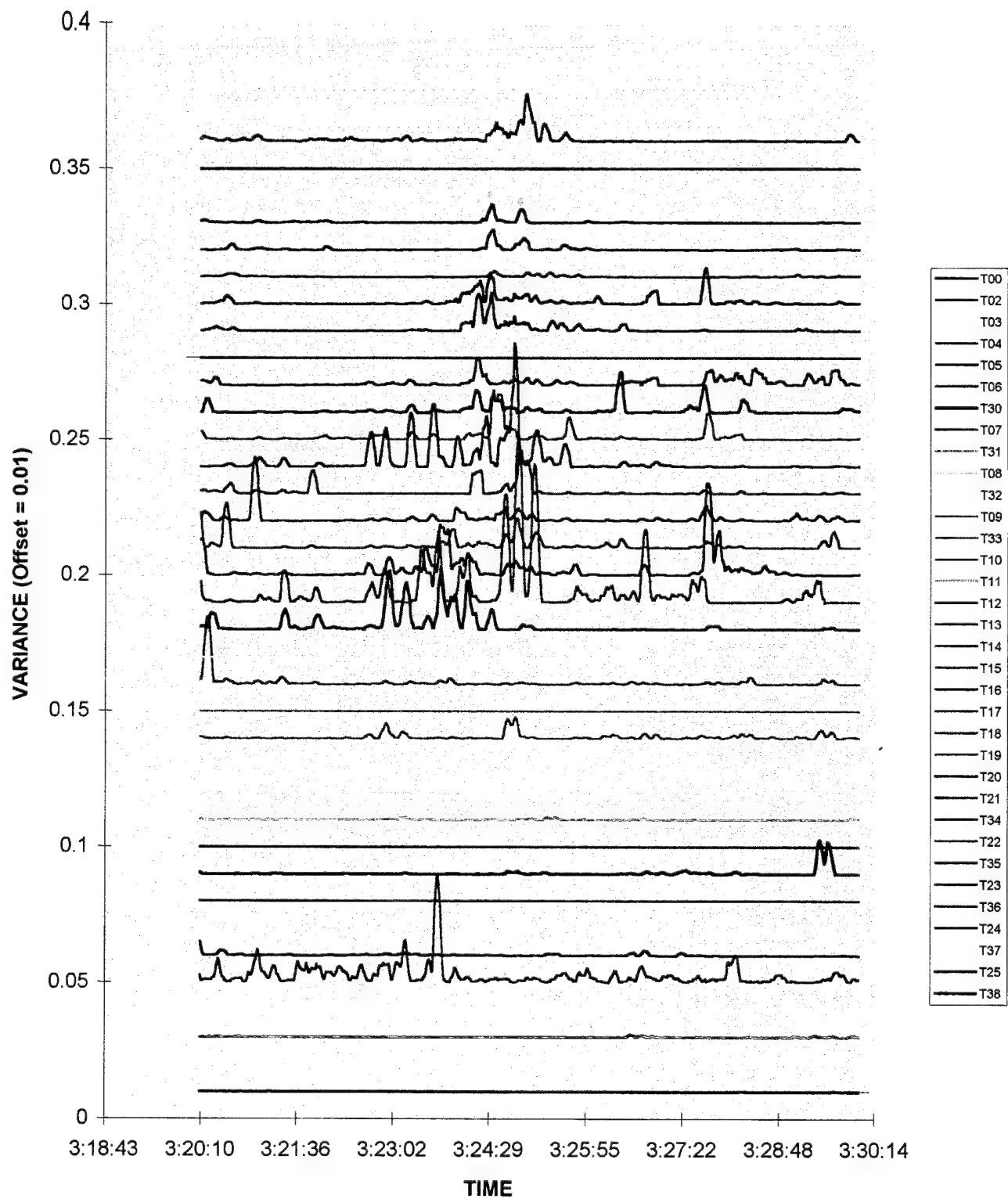
TEMPERATURE DATA 09/14/94 (03:20 - 03:30)



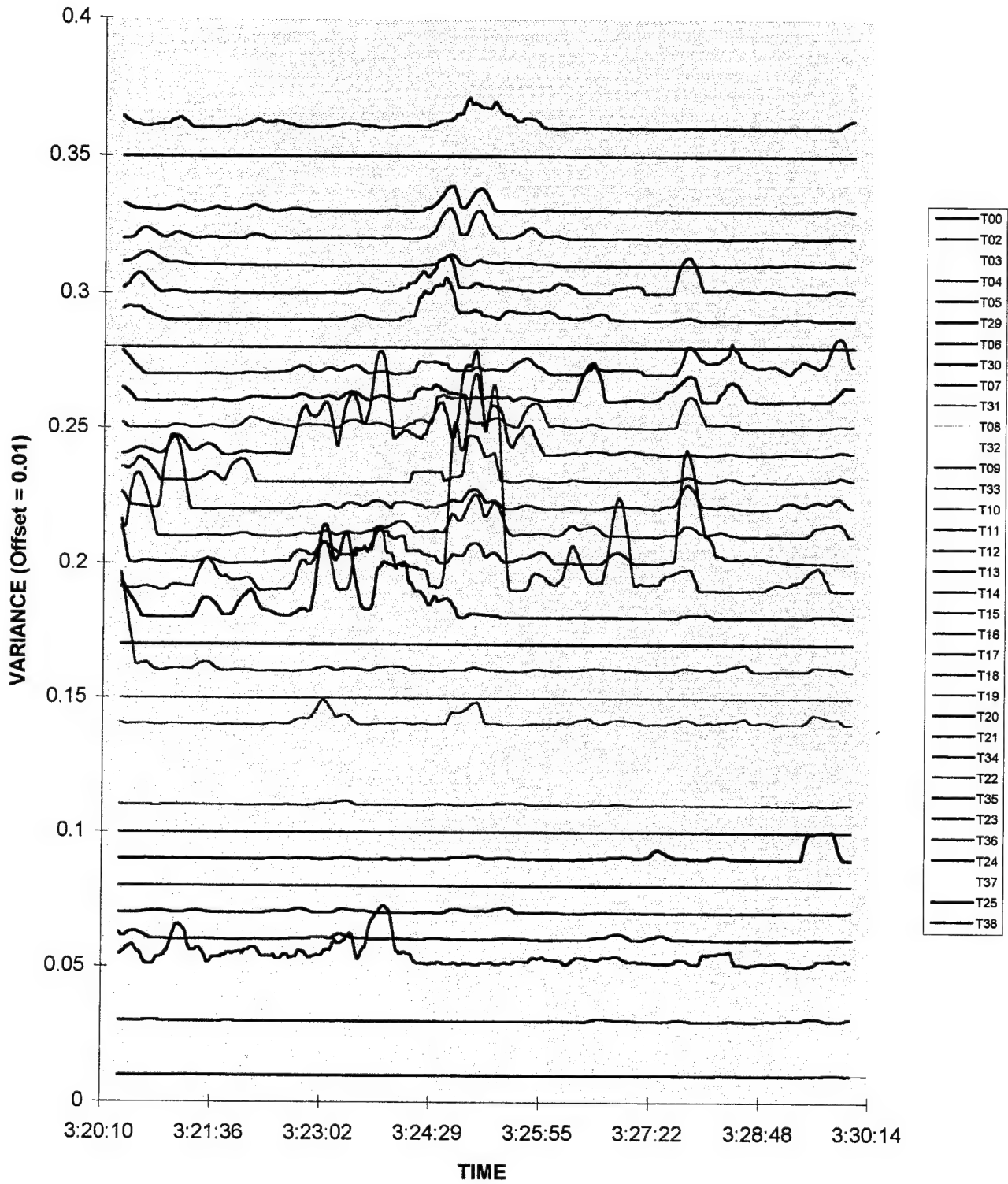
TEMPERATURE VARIANCE (5 Point Sample)



TEMPERATURE VARIANCE (10 Point Sample)



TEMPERATURE VARIANCE (25 Point Sample)



Appendix A. Software Documentation

/*Header file used in the data processing and database functions.

Must be included in each function (10/18/94 - RKM)*/

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
```

```
#define TRUE 1
#define FALSE 0
```

```
struct TssDataStruct
{
    char Date[9];
    char Time[9];
    float Lon;
    float Lat;
    float Pres;
    float Temp[39];          /*temp array*/
    float Cond[3];           /*conductivity array*/
    float Bstr[2];           /*backscatter array*/
    float Flor[8];           /*flourometer array*/
    float Irr[4];            /*irradiance array*/
    float Sirr[4];           /*surface irradiance array*/
    float Gmux;
    float Vref;
    float Roll;
    float Pitch;
    float Tnsn;
    float Clamp;
    float C28V;
};
```

```
/*Function Prototypes*/
void main();
void InitFiles();
void PositionOut(struct TssDataStruct *);
void TempOut(struct TssDataStruct *);
void CondOut(struct TssDataStruct *);
void OpticsOut(struct TssDataStruct *);
void FlourOut(struct TssDataStruct *);
```

```
void ReferenceOut(struct TssDataStruct *);  
void DbPopulate(char *);
```

```
/*Main function for parts program, starts processing and issues instructions  
on programs use and operation. (10/18/94 - RKM)*/
```

```
#include <parts.h>
```

```
void main()
```

```
{
```

```
    int ch;
```

```
    system ("clear");
```

```
    printf ("\nWELCOME TO PARTS - Processing And Retrieval of Towed Sensor data\n\n");
```

```
    printf (" This program will create an Ingres database using the named\n");
```

```
    printf (" Towed Sensor System (TSS) data file as input. The program will\n");
```

```
    printf (" request the data file name and the name of the target database.\n\n");
```

```
    printf (" Return to start or q to quit: ");
```

```
    ch = getchar();
```

```
    if ((ch == 'q') || (ch == 'Q'))
```

```
    {
```

```
        system ("clear");
```

```
        exit(0);
```

```
    }
```

```
    else
```

```
    {
```

```
        /*Request files names and open necessary files*/
```

```
        InitFiles();
```

```
        system ("clear");
```

```
        printf ("\n FINISHED PROCESSING TSS DATA\n\n");
```

```
    }
```

```
}
```

/*Function to read raw (1 sec) Tss data file and produces Ingres input files,
Each line of the 1 sec file is read using formatted input (scanf). Individual
sensor values are then transferred to the appropriate function for further
processing and output into individual sensor data files. (10/18/94 - RKM)*/

```
#include <parts.h>
#include <unistd.h>

void InitFiles()
{
    /*1 sec data file*/
    FILE *TssRaw;

    struct TssDataStruct TssData;
    char Infile[256], DbName[256], ch;
    int i;

    system ("clear");
    printf ("\n\nPARTS - Processing And Retrieval of Towed Sensor data\n\n");
    /*prompt for file name*/
    printf (" Enter input TSS Data Filename: ");
    scanf ("%s", Infile);

    /*prompt for Ingres database name, by default data base is created in the
    /home/ingres/data/base directory*/
    printf ("\n Enter target INGRES database name: ");
    scanf ("%s", DbName);

    system ("clear");
    printf ("\n\nPROCESSNG\n\n");
    printf (" Input 1 second file = %s\n", Infile);
    printf (" Ingres database name = %s\n", DbName);

    /*Create Ingres database*/
    if (fork() == 0)
        execlp ("creatdb", "creatdb", DbName, 0);
    else
    {
        printf ("ERROR: Cannot create Ingres database\n");
        exit(1);
    }

    /*Open 1 second data file for ouput*/
```

```

TssRaw = fopen (Infile, "r");
if (TssRaw == NULL)
{
    printf ("ERROR - CANNOT OPEN INPUT FILE\n");
    exit(1);
}

/*Discard 1 second file header line*/
while ((ch = fgetc (TssRaw)) != '\n');

/*Read and process each line and create Ingres data files*/
while ((fscanf (TssRaw, "%s%s%f%f%f", TssData.Date, TssData.Time,
    &TssData.Lon, &TssData.Lat, &TssData.Pres)) != EOF)
{
    /*Read the temperature values*/
    for (i=0; i<39; i++)
        fscanf (TssRaw, "%f", &TssData.Temp[i]);

    /*Read conductivity values*/
    for (i=0; i<3; i++)
        fscanf (TssRaw, "%f", &TssData.Cond[i]);

    /*Read backscatter values*/
    for (i=0; i<2; i++)
        fscanf (TssRaw, "%f", &TssData.Bstr[i]);

    /*Read flourmeter values*/
    for (i=0; i<8; i++)
        fscanf (TssRaw, "%f", &TssData.Flor[i]);

    /*Read irradiance values*/
    for (i=0; i<4; i++)
        fscanf (TssRaw, "%f", &TssData.Irr[i]);

    /*Read reference data values*/
        fscanf (TssRaw, "%f%f%f%f", &TssData.Gmux, &TssData.Vref, &TssData.Roll,
&TssData.Pitch);
        fscanf (TssRaw, "%f%f%f%f", &TssData.Tnsn, &TssData.Clamp, &TssData.C28V,
&TssData.Sirr[0]);

    /*Read surface irradiance values*/
    fscanf (TssRaw, "%f%f%f%f\n", &TssData.Sirr[1], &TssData.Sirr[2], &TssData.Sirr[3]);

```

```
/*Functions to create files for input to Ingres database*/
PositionOut (&TssData);
TempOut (&TssData);
CondOut (&TssData);
OpticsOut (&TssData);
FlourOut (&TssData);
ReferenceOut (&TssData);
}

/*Function to populate the Ingres database*/
DbPopulate(DbName);
}
```

/*Function to create position data file for input to Ingres Database.
File is called position.dat and is created in the directory from
which the parts program starts. This data file contains the date,
time, latitude and longitude of each data point in the database.
(10/19/94 - RKM)*/

```
#include <parts.h>
```

```
void PositionOut (struct TssDataStruct *pTssData)
```

```
{  
    FILE *PositionFile;
```

```
    PositionFile = fopen ("position.dat", "a");
```

```
    if (PositionFile == NULL)
```

```
    {  
        printf ("ERROR: Cannot open position file\n");  
        exit (1);  
    }
```

```
    /*Write position data to file*/
```

```
    fprintf (PositionFile, "%s\t%s\t%f\t%f\n", pTssData->Date, pTssData->Time,  
            pTssData->Lon, pTssData->Lat);
```

```
    fclose (PositionFile);
```

```
}
```

/*Function to create temperature data file for input to Ingres Database
File is called temp.dat and is created in the deirectory form which the
parts program starts. This data file contains the date, time, depth and
temperature value of each sensor. (10/20/94 - RKM)

Sensor positions are relative to pressure sensor, this value must be subtracted
from the location of the sensor to obtain the actual depth of the sensor at the
time the temperature was noted.*/

#include <parts.h>

void TempOut (struct TssDataStruct *pTssData)

```
{  
    FILE *TempFile;  
  
    /*Structure Containing Temp Sensor ID and Locations*/  
    struct TempInfoStruct  
    {  
        char    Id[4];  
        float   Pos;  
    }TempInfo[39];
```

```
    int i;  
    float depth;
```

```
    /*Open temperature data file for input to Ingres*/  
    TempFile = fopen ("temp.dat", "a");  
    if (TempFile == NULL)  
    {  
        printf ("ERROR: Cannot open temperature file\n");  
        exit(1);  
    }
```

```
    /*Initialize TempInfo, .Pos gives each sensors relative position to the  
    pressure sensor, this value is unique and must be subtracted from the pressure  
    value to obtain depth*/
```

```
    strcpy (TempInfo[0].Id, "T00");  
    TempInfo[0].Pos = 11.16;
```

```
    strcpy (TempInfo[1].Id, "T01");  
    TempInfo[1].Pos = 13.19;
```

```
    strcpy (TempInfo[2].Id, "T02");
```

```
TempInfo[2].Pos = 14.22;

strcpy (TempInfo[3].Id, "T03");
TempInfo[3].Pos = 17.26;

strcpy (TempInfo[4].Id, "T04");
TempInfo[4].Pos = 19.29;

strcpy (TempInfo[5].Id, "T05");
TempInfo[5].Pos = 21.32;

strcpy (TempInfo[6].Id, "T29");
TempInfo[6].Pos = 22.34;

strcpy (TempInfo[7].Id, "T06");
TempInfo[7].Pos = 23.35;

strcpy (TempInfo[8].Id, "T30");
TempInfo[8].Pos = 24.37;

strcpy (TempInfo[9].Id, "T07");
TempInfo[9].Pos = 25.38;

strcpy (TempInfo[10].Id, "T31");
TempInfo[10].Pos = 26.40;

strcpy (TempInfo[11].Id, "T08");
TempInfo[11].Pos = 27.42;

strcpy (TempInfo[12].Id, "T32");
TempInfo[12].Pos = 28.43;

strcpy (TempInfo[13].Id, "T09");
TempInfo[13].Pos = 29.45;

strcpy (TempInfo[14].Id, "T33");
TempInfo[14].Pos = 30.46;

strcpy (TempInfo[15].Id, "T10");
TempInfo[15].Pos = 31.48;

strcpy (TempInfo[16].Id, "T11");
TempInfo[16].Pos = 32.50;
```

strcpy (TempInfo[17].Id, "T12");
TempInfo[17].Pos = 33.51;

strcpy (TempInfo[18].Id, "T13");
TempInfo[18].Pos = 34.53;

strcpy (TempInfo[19].Id, "T14");
TempInfo[19].Pos = 35.54;

strcpy (TempInfo[20].Id, "T15");
TempInfo[20].Pos = 35.56;

strcpy (TempInfo[21].Id, "T16");
TempInfo[21].Pos = 37.58;

strcpy (TempInfo[22].Id, "T17");
TempInfo[22].Pos = 38.59;

strcpy (TempInfo[23].Id, "T18");
TempInfo[23].Pos = 39.61;

strcpy (TempInfo[24].Id, "T19");
TempInfo[24].Pos = 40.62;

strcpy (TempInfo[25].Id, "T20");
TempInfo[25].Pos = 41.13;

strcpy (TempInfo[26].Id, "T21");
TempInfo[26].Pos = 41.64;

strcpy (TempInfo[27].Id, "T34");
TempInfo[27].Pos = 42.15;

strcpy (TempInfo[28].Id, "T22");
TempInfo[28].Pos = 42.66;

strcpy (TempInfo[29].Id, "T35");
TempInfo[29].Pos = 43.16;

strcpy (TempInfo[30].Id, "T23");
TempInfo[30].Pos = 43.77;

strcpy (TempInfo[31].Id, "T36");

```

TempInfo[31].Pos = 44.18;

strcpy (TempInfo[32].Id, "T24");
TempInfo[32].Pos = 44.69;

strcpy (TempInfo[33].Id, "T37");
TempInfo[33].Pos = 45.20;

strcpy (TempInfo[34].Id, "T25");
TempInfo[34].Pos = 45.70;

strcpy (TempInfo[35].Id, "T38");
TempInfo[35].Pos = 46.72;

strcpy (TempInfo[36].Id, "T26");
TempInfo[36].Pos = 47.74;

strcpy (TempInfo[37].Id, "T27");
TempInfo[37].Pos = 48.75;

strcpy (TempInfo[38].Id, "T28");
TempInfo[38].Pos = 50.28;

/*output temp data and calculate "true depth"*/
for (i=0; i<39; i++)
{
    depth = pTssData->Pres - TempInfo[i].Pos;

    /*Remove any "wild" values and flag with 99.9*/
    if (pTssData->Temp[i] < -2.0 || pTssData->Temp[i] > 40.00)
        pTssData->Temp[i] = 99.9;

    fprintf (TempFile, "%s\t%s\t%s\t%f\t%f\n", pTssData->Date, pTssData->Time,
            TempInfo[i].Id, pTssData->Temp[i], depth);
}
fclose (TempFile);
}

```

```

/*Create file containing conductivity data for input to Ingres Database.
File is called position.dat and is created in the directory from
which the parts program starts. This data file contains the date,
time, conductivity value and actual depth calculated from pressure
and sensor position, for each conductivity sensor. (10/21/94 - RKM)*/

```

```

#include <parts.h>

```

```

void CondOut (struct TssDataStruct *pTssData)
{
    FILE *CondFile;

```

```

/*Structure containing Conductivity sensor ID and location*/
struct CondInfoStruct
{
    char      Id[4];
    float     Pos;
}CondInfo[3];

```

```

int i;
float depth;

```

```

CondFile = fopen ("cond.dat", "a");
if (CondFile == NULL)
{
    printf ("ERROR: Cannot open conductivity file\n");
    exit(1);
}

```

```

/*Initialize CondInfo, .Pos hold the sensor position relative to the
pressure sensor, this value is used to calculate sensor depth*/
strcpy (CondInfo[0].Id, "C01");
CondInfo[0].Pos = 42.15;

```

```

strcpy (CondInfo[1].Id, "C02");
CondInfo[1].Pos = 43.16;

```

```

strcpy (CondInfo[2].Id, "C03");
CondInfo[2].Pos = 44.18;

```

```

/*Output conductivity data and calculate "actual depth"*/
for (i=0; i<3; i++)
{

```

```
depth = pTssData->Pres - CondInfo[i].Pos;
fprintf (CondFile, "%s\t%s\t%s\t%f\t%f\n", pTssData->Date, pTssData->Time,
        CondInfo[i].Id, pTssData->Cond[i], depth);
}
fclose (CondFile);
}
```

/*Function to create optical data data file for input to Ingres database.

File is called optics.dat and is created in the directory from which the parts program starts. This data file contains the date, time, and all optical data. (10/20/94 - RKM)

Combined file of all TSS optical data: Backscatter(Bstr), Subsurface-irradiance (Irr) and surface-irradiance (Sirr).

Sensor positions are relative to the pressure sensor and must be subtracted to obtain actual depth*/

#include <parts.h>

void OpticsOut (struct TssDataStruct *pTssData)

{
FILE *OpticsFile;

/*Structure containing backscatter sensor IDs and locations*/

struct BstrInfoStruct

{
char Id[5];
float Pos;
}BstrInfo[2];

/*Structure containing subsurface irradiance IDs and locations*/

struct IrrInfoStruct

{
char Id[5];
float Pos;
}IrrInfo[4];

/*Structure containing surface irradiance IDs*/

struct SirrStruct

{
char Id[4];
}SirrInfo[4];

int i;

float depth;

OpticsFile = fopen ("optics.dat", "a");

if (OpticsFile == NULL)

{

```

printf ("ERROR: Cannot open optics data file\n");
exit(1);
}

```

```

/*Initialize BstrInfo, .Pos contains the sensor position relative to the
pressure sensor and is subtracted to obtain depth*/
strcpy (BstrInfo[0].Id, "Bs01");
BstrInfo[0].Pos = 18.98;

```

```

strcpy (BstrInfo[1].Id, "Bs02");
BstrInfo[1].Pos = 41.84;

```

```

/*Initialize IrrInfo, .Pos contains the sensor position relative to the
pressure sensor and is subtracted to obtain depth*/
strcpy (IrrInfo[0].Id, "Ir01");
IrrInfo[0].Pos = 11.16;

```

```

strcpy (IrrInfo[1].Id, "Ir02");
IrrInfo[1].Pos = 25.38;

```

```

strcpy (IrrInfo[2].Id, "Ir03");
IrrInfo[2].Pos = 35.54;

```

```

strcpy (IrrInfo[3].Id, "Ir04");
IrrInfo[3].Pos = 50.28;

```

```

/*Initialize SIRRId, surface irradiance is measured at the surface using a
deck mounted sensor*/
strcpy (SirrInfo[0].Id, "410");
strcpy (SirrInfo[1].Id, "488");
strcpy (SirrInfo[2].Id, "550");
strcpy (SirrInfo[3].Id, "683");

```

```

/*Output backscatter data and calculate "actual depth"*/
for (i=0; i<2; i++)
{
    depth = pTssData->Pres - BstrInfo[i].Pos;
    fprintf (OpticsFile, "%s\t%s\t%s\t%f\t%f\n", pTssData->Date, pTssData->Time,
            BstrInfo[i].Id, pTssData->Bstr[i], depth);
}

```

```

/*Output subsurface irradiance data and calculate "actual depth"*/
for (i=0; i<4; i++)

```

```

{
    depth = pTssData->Pres - IrrInfo[i].Pos;
    fprintf (OpticsFile, "%s\t%s\t%s\t%f\t%f\n", pTssData->Date, pTssData->Time,
            IrrInfo[i].Id, pTssData->Irr[i], depth);
}

/*Output surface irradiance data and IDs*/
for (i=0; i<4; i++)
{
    depth = 0;    /* surface measurment*/
    fprintf (OpticsFile, "%s\t%s\t%s\t%f\t%f\n", pTssData->Date, pTssData->Time,
            SIRRInfo[i].Id, pTssData->SIRR[i], depth);
}

fclose (OpticsFile);
}

```

/*Function to create flourometer data file for input to Ingres database.
File is called flour.dat and is created in the directory from
which the parts program starts. This data file contains the date,
time, depth and flourometer sensor data.

Depth is determined by subtracting the sensor position from the pressure
obtained by the pressure sensor. (10/21/95 - RKM)*/

```
#include <parts.h>
```

```
void FlourOut (struct TssDataStruct *pTssData)
```

```
{  
    FILE *FlourFile;
```

```
    /*structure containing flourometer sensor ID and positions*/
```

```
    struct FlourInfoStruct
```

```
{  
    char Id[5];  
    float      Pos;  
}FlourInfo[8];
```

```
int i;  
float depth;
```

```
FlourFile = fopen ("flour.dat", "a");
```

```
if (FlourFile == NULL)
```

```
{  
    printf ("ERROR: Cannot open flourometer file\n");  
    exit(1);  
}
```

```
/*Initialize FlourInfo, .Pos contains the location of the sensor, the depth is  
determined by subtracting the location from the pressure sensor value.*/
```

```
strcpy (FlourInfo[0].Id, "FI01");
```

```
FlourInfo[0].Pos = 39.51;
```

```
strcpy (FlourInfo[1].Id, "FI02");
```

```
FlourInfo[1].Pos = 40.52;
```

```
strcpy (FlourInfo[2].Id, "FI03");
```

```
FlourInfo[2].Pos = 41.54;
```

```
strcpy (FlourInfo[3].Id, "FI04");
```

```

FlourInfo[3].Pos = 42.55;

strcpy (FlourInfo[4].Id, "Fl05");
FlourInfo[4].Pos = 43.57;

strcpy (FlourInfo[5].Id, "Fl06");
FlourInfo[5].Pos = 44.59;

strcpy (FlourInfo[6].Id, "Fl07");
FlourInfo[6].Pos = 45.60;

strcpy (FlourInfo[7].Id, "Fl08");
FlourInfo[7].Pos = 46.62;

/*Output flourometer data and calculate "actual depth"*/
for (i=0; i<8; i++)
{
    depth = pTssData->Pres - FlourInfo[i].Pos;
    fprintf (FlourFile, "%s\t%s\t%s\t%f\t%f\n", pTssData->Date, pTssData->Time,
            FlourInfo[i].Id, pTssData->Flor[i], depth);
}

fclose (FlourFile);
}

```

```

/*Function to create reference data file for input to Ingres database
File is called ref.dat and is created in the directory from
which the parts program starts. This data file contains the date,
time and reference data collected for each data point.
(10/21/94 - RKM)*/

```

```

#include <parts.h>

```

```

void ReferenceOut (struct TssDataStruct *pTssData)

```

```

{
    FILE *RefFile;

    RefFile = fopen ("ref.dat", "a");
    if (RefFile == NULL)
    {
        printf ("ERROR: Cannot open reference data file\n");
        exit(1);
    }

```

```

/*Output reference data:

```

```

    Gmux: multiplexer noise level,
    Vref: DC reference voltage,
    Roll: Measure of depressor roll in degrees,
    Pitch: Depressor pitch in degrees,
    Tension: Cable tension measured at the depressor,
    Clamp: Rectified AC voltage,
    C28volts: DC voltage */
    fprintf (RefFile, "%s\t%s\t%f\t%f\t%f\t%f\t%f\t%f\n", pTssData->Date,
        pTssData->Time, pTssData->Gmux, pTssData->Vref, pTssData->Roll,
        pTssData->Pitch, pTssData->Tnsn, pTssData->Clamp, pTssData->C28V);

    fclose (RefFile);
}

```

```

/*Create relations in Ingres database and populate with data generated from
previous processing. This function contains embedded Ingres call which create
the relations in the chosen database. Data files are removed as their relations
are created in order to maximize disk space usage.(10/25/94)*/

```

```

char DataBase[256];

```

```

void DbPopulate(char *DbName)

```

```

{

```

```

    strcpy (DataBase, DbName);

```

```

/*Open Ingres database for transactions*/

```

```

{Iingres(DataBase,0);}

```

```

/*Create and populate position data relation*/

```

```

{Iwrite("create position(date=c8,time=c8,lon=c10,lat=c10)");

```

```

Iisync(0);} {Iwrite("copy position(date=c0,time=c0");

```

```

Iwrite(",lon=c0,lat=c0)from\"/home/rick/source/PARTS/position.dat\"");Iisync(0);}

```

```

/*Remove position data file*/

```

```

system ("delete.position");

```

```

/*Create and populate temperature data relation*/

```

```

{Iwrite("create temp(date=c8,time=c8,id=c3,val=f4,depth=f4)");

```

```

Iisync(0);} {Iwrite("copy temp(date=c0,time=c0,id");

```

```

Iwrite("=c0,val=c0,depth=c0)from\"/home/rick/source/PARTS/temp.dat\"");Iisync(0);}

```

```

/*Remove temperature data file*/

```

```

system ("delete.temp");

```

```

/*Create and populate conductivity data relation*/

```

```

{Iwrite("create cond(date=c8,time=c8,id=c3,val=f4,depth=f4)");

```

```

Iisync(0);} {Iwrite("copy cond(date=c0,time=c0,id");

```

```

Iwrite("=c0,val=c0,depth=c0)from\"/home/rick/source/PARTS/cond.dat\"");Iisync(0);}

```

```

/*Remove conductivity data file*/

```

```

system ("delete.cond");

```

```

/*Create and populate optics data relation*/

```

```

{Iwrite("create optics(date=c8,time=c8,id=c4,val=f4,depth=f4)");

```

